Curl

A Gentle Slope Language for the Web

M. Hostetter, D. Kranz, C. Seed, C. Terman, S. Ward

Abstract

The work described in this paper explores the design of Curl, a single, coherent linguistic basis for expression of Web content at levels ranging from simple formatted text to contemporary object-oriented programming. Curl is part of a research effort aimed at eliminating discontinuities from the function/sophistication curve. This yields an environment in which (1) incremental functionality requires incremental skill acquisition, and (2) a consistent semantics avoids communication obstacles between separately encapsulated fragments of content. We characterize Curl as a "gentle slop esystem" because it makes it easy to transition from one point to another in the function/sophistication spectrum.

Introduction

Curl is a new language for creating Web documents with almost any sort of content, from simple formatted text to complex interactive applets.

- *Text formatting.* Curl provides a rich set of formatting operations similar to those implemented by HTML tags. Unlike HTML, the Curl formatter can be extended by users to provide additional functionality, from simple macros (to provide a convenient way to switch to a particular font, size, and color) to direct control over the positioning of subcomponents (as in a TeX-like equation formatter). Several packages of useful formatting extensions are currently under development.

- *Scripting simple interactions.* Using a TK-like interface toolkit of interactive components, Curl makes it easy to build simple interactive Web pages. One can view interactive objects like buttons or editable fields as extensions to the basic formatting operations provided above--one uses the same easy-to-learn syntax to create interactive documents as to create regular text documents. There's no need to learn a separate scripting language!

- *Programming complex operations.* Other components of an interactive document may require more sophisticated mechanisms than are provided by the interface toolkit. These components can also be developed using Curl since, at its heart, Curl is really an object-oriented programming language. Curl expressions, class definitions, and procedure definitions embedded in the Web document are securely compiled to native code by the built-in, on-the-fly compiler and then executed without the need for any sort of interpreter. Curl provides many of the features of a modern object-oriented programming language: multiple inheritance, extensible syntax, a strong type system that includes a dynamic "any" type, safe execution through encapsulation of user code, and extensive checking performed both at compile and run time. Almost all of the Curl system and compiler are written in Curl.

Curl is intended to be a *gentle slope system*, accessible to content creators at all skill levels, ranging from authors new to the Web to experienced programmers. By using a simple, uniform language syntax and semantics, Curl avoids the discontinuities experienced by current Web users who have to juggle HTML, JavaScript, Java, Perl, etc. to create today's exciting sites. Our hope is that the single environment provided by Curl will be an attractive alternative for Web developers.

Gentle Slope Systems

Is a new language for specifying the content of Web documents really necessary? We think the answer is "yes" because of the following:

- The changing role of the Web

- The wide range of skills present in the community of Web authors

The World Wide Web is evolving from the constrained goal of hypertext presentation to a much more demanding role in networked computing: it is becoming the standard user interface to a wide and potentially unbounded class of computational services. This evolution naturally strains the capabilities of the simple markup language on which most Web content has been based. In order to accommodate new user-interaction models and functionality, the simple formatting functions of HTML [6,7,8] are commonly augmented by applets and scripts based on more powerful linguistic tools such as JavaScript [1], Java [2,3], or C++ [16]. Curl is designed to have the positive features of markup languages, scripting languages, and statically typed object-oriented programming languages.

BEST AVAILABLE COPY

This complex of independently conceived content production tools confronts potential Web authors with annoying--and potentially crippling--semantic and performance discontinuities. The enhancement of a Web page to include, say, an animation or interactive simulation requires escape from the simple markup language to an unrelated programming language within which the new functionality may be encapsulated. Such discontinuities are undesirable for at least two reasons:

1. They represent technological barriers that require substantial new skills to overcome: they tend to partition the content creators into nearly disjoint sets of *authors* versus *programmers*.

2. They present serious interface constraints among separately encapsulated functions. It is difficult, for example, to reference some result of an encapsulated computation from external hypertext. It is also difficult to share structured data between different language levels because they usually use different representations.

Curl avoids these problems by using a consistent syntax and semantics for the expression of Web content at levels ranging from simple formatted text to contemporary object-oriented programming.

Thus, authors with simple tasks can write straightforward code and add more details when more complicated functionality or performance is required. We characterize Curl as a "gentle slope system" because it makes it easy to transition from one point to another in the function/sophistication spectrum.

Technical challenges confronted in the design of such a language stem largely from tensions between the ease-of-use characteristics we demand of formatting and scripting languages, and the performance and expressiveness goals we expect of contemporary system programming languages. Strong compile-time typing, for example, is inhumane in the former but essential in the latter. Storage allocation, object orientation, and the handling of English textual content are further examples where the tension between ease-of-use and performance/expressiveness present interesting conundrums for the language designer. In most cases where compromise was necessary, the scripting language aspects were treated as most important.

More generally, we are anxious to capture in Curl the easy approachability to which the Web owes much of its popularity. To a great extent this approachability derives from careful design decisions that allow (or even require) *under-specification* of function (e.g., output format), in contrast to conventional programming language semantics geared to the complete specification of behavior. Curl's design is explicitly aimed at reconciling the spirit of under-specification with the requirement that it scale to the sophisticated end of the programming spectrum.

Curl Overview

The name *Curl* derives from the principal syntactic feature of the language: Curl source files are arbitrary text, punctuated by expressions enclosed by curly brackets ({}). Like HTML, plain text is valid source; unlike HTML, the escapes extend to a real programming language. (See Figure 1.)

Interpreting the Curl source in Figure 1 requires definitions for the `bold` and + operators. In Curl, the meaning of `{operator ...}` is determined by first locating a definition for `operator` in the current environment and then calling upon that function to parse and interpret the remaining text ". . . " of the expression. Note that this mechanism allows the ". . ." text to have any desired structure since the parsing rules are dynamically defined.

In this example, the `bold` operator treats its arguments as a sequence of `Graphic` objects (in this case simple characters) to be displayed with the `'bold` property set to true, while the + operator parses the remaining text as a sequence of subexpressions to be evaluated and then summed. Every Curl object can be converted to a Graphic object (the conversion process is controlled by a method defined by the object's type) which, in turn, can be assembled into a formatted display by the Curl browser. An interesting side-effect of being able to view all Curl objects is that we can use the browser metaphor to good effect when building debuggers, inspectors, etc.

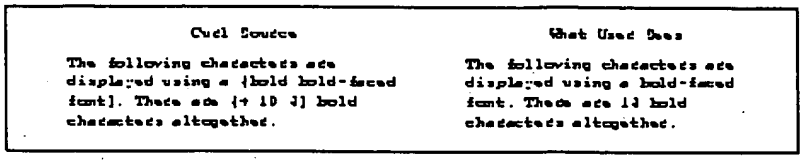
Curl comes with a large (and growing) repertoire of predefined operators which provide formatting operations and a TK-like toolkit of user-interface components. The widespread adoption of TCL/TK and, to a lesser extent, JavaScript/HTML, demonstrates that simple applications of these operators are often sufficient to fulfill many, if not most, of the needs of interactive documents. Figure 2 assembles a few of these components into a simple order form.

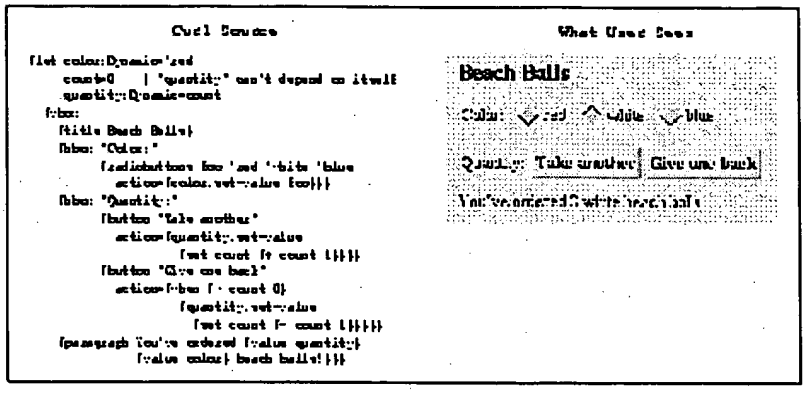
Since the values for `color` and `quantity` are `Dynamic` objects, the last line of the display changes automatically as the user manipulates the color and quantity controls. A `Dynamic` object incorporates a simple mechanism for propagating changes in its value to other dynamic objects that depend on first object's value. More sophisticated propagation rules could be supplied by the user by creating a new class of objects derived from `Dynamic` objects that have a different "propagate" method.

The screen shot in Figure 2 reflects the fact the user has selected something besides the default color (red) and quantity (0). Note that the "text-as-program paradigm" means that the `value` operator must be wrapped around variable names inside of paragraphs. Otherwise, the

name of the variable as a text string will be displayed instead of its value. This example illustrates several Curl features:

- *Formatting model.* A TeX-like hierarchical "box-and-glue" formatting model that makes it easy to construct layouts.

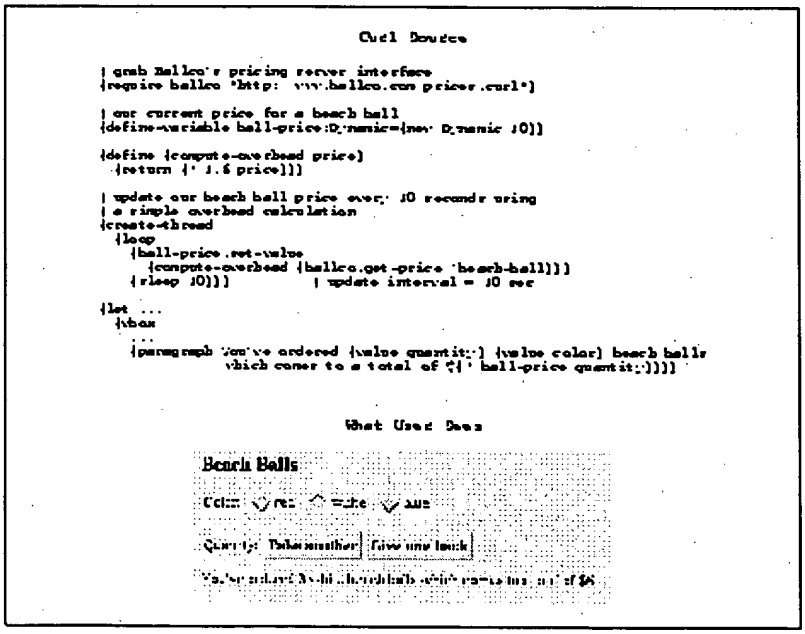




- *Naming environment.* A lexically-scoped naming environment which allows one to construct modular documents with interchangeable components. Languages with "flat" name spaces, like TCL, can make it difficult to reuse components since naming conflicts need to be resolved as code fragments are moved from document to document.

- *Object-oriented programming.* The use of objected-oriented programming to extend the behavior of the Curl system in an easily-accessible way. In this example the creator of the `Dynamic` class has encapsulated the knowledge of how a change in the value of a `Dynamic` object results ultimately in the updating of the display. Of course the current user doesn't have to understand how the underlying mechanism works in order to put it to effective use. This sort of extensibility is beyond the ken of most simple scripting languages.

Now suppose one wanted to compute the sales price too.



The Curl source in Figure 3 starts by reading in another Curl document into an environment named "Ballco." Environments completely

encapsulate a document, making it easy to implement various security, storage management, and clean-up policies. Environments are themselves objects with instance variables and methods derived from the top-level definitions in the source documents. Thus `ballco.get-price` is a reference to the `get-price` procedure defined in Ballco's *prices.curl* file.

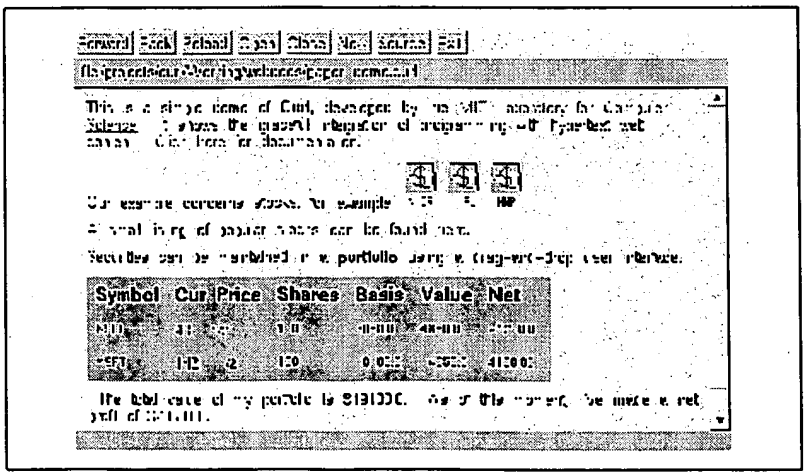
This example also illustrates the use of additional *threads of control* to create independent tasks that run in parallel with the user's main computation. In this case a separate thread is given the task of periodically querying the Ballco database for the latest beach ball price which is then used (with a suitable markup!) to update the price in the example document. Curl threads are preemptive and not specific to any particular platform; a simple `lock` object with `acquire` and `release` methods is used as the lowest-level synchronization primitive. The combination of threads and dynamic values makes it simple to incorporate information gleaned from the Web into a document. It also shows how procedure and variable definitions are simply part of the document, though not displayed.

It is easy to see how a document can grow in incremental steps from a simple text file to include ever more sophisticated interface and information-gathering components. To make this transition a natural one, Curl provides a whole spectrum of operators. Of course, users are likely to want operations we haven't thought of (or had time to implement) so it makes sense to provide a way for new operators to be added to the Curl environment. The Curl graphical interface supports the usual things that Web users expect such as tables, hyperlinks, and images.

Example 1 shows a simple drag-and-drop interface and how a more sophisticated programmer can extend the functions available to technically naive authors. It is a Web page, designed by a Curl neophyte, that uses a `portfolio` object to track investments.

```
{require portfolio "http://www.portfolios-galore.com/portfolio.curl"}

This is a simple demo of {bold Curl}, developed by the
{anchor url="http://cag-www.lcs.mit.edu" MIT Laboratory for Computer Science}.
It shows the graceful integration of programming with hypertext
Web content.  {anchor url="Documentation/overview.curl" Click here} for
documentation.

Our example concerns stocks, for example:
{portfolio.stock "NWIR" shares=500 cost=16.125}
{portfolio.stock "AAPL" shares=500 cost=23.5}
{portfolio.stock "HWP" shares=500 cost=48.25}

A small listing of popular stocks can be found {anchor url="topstocks.curl" here}.

Securities can be maintained in a {bold portfolio} using a drag-and-drop
user interface.

{let p={new portfolio.PortfolioBox
          {portfolio.stock "NSCP" units=100  cost=60.5}
          {portfolio.stock "MSFT" units=100 cost=101}
          }
  {vbox p.graphic
       {paragraph
       The total value of my portfolio is ${value p.value}.  As of this
       moment, I've made a net profit of ${- p.value p.cost}.}}}
```

Figure 4 shows the document/program in Example 1 viewed through the Curl browser.



`PortfolioBox` and `stock` have been imported from the portfolio URL and offer some sophisticated features. The `PortfolioBox` displays as a table and accesses a quote server to maintain up-to-date prices. It provides direct-access editing features for portfolio manipulation, including a simple drop method that allows the shown stock objects to be dragged onto the portfolio window to automatically extend the set of securities tracked. The `stock` object has a simple drag method. In Figure 3, the `value` and `cost` fields of the portfolio are Dynamic, and the last paragraph will be automatically updated as the quantities of stock or their values change.

Curl as a Programming Language

Curl is both a language and an authoring environment, where the authoring environment was implemented as user extensions to Curl. The language can be used as an HTML replacement for the presentation of formatted text, whose capabilities include those of scripting languages as well as compiled, strongly-typed, object-oriented system programming.

Obvious influences on Curl's design include LISP [4,13,14,15], C++, Tcl/Tk [5], TeX [9], and HTML. Language design decisions directly reflect the goal of reconciling its simple accessibility to naive users with the power and efficiency demanded by sophisticated programming tasks. Salient features of Curl include the following:

Programs as documents

Execution of every Curl program results in a document containing a set of objects that may be displayed or otherwise manipulated. Curl includes an extensible GUI which provides for simple, structured arrangement of displayable objects. The debugging and development environment is integrated and accessible. Curl objects, along with the type information that describes them, may be browsed along with other Web content. Objects seen on the screen may be identified and inspected.

Syntax and Semantics

Extensibility. Arbitrary sublanguages can be embedded as new {. . .} forms and can be processed correctly by Curl, by simple extension of the environment. For example, the following Curl form:

```
{circuit-simulation-netlist
  M1 out in gnd gnd nmos w=4u l=1u
  M2 out in vdd vdd pmos w=8u l=1u
  VDD vdd gnd 5v
  VIN in gnd pwl(0 0 0.1 5)
  .tran .1n 10n
  .plot v(in) v(out)}
```

might be found in a document for an engineering course where the instructor wished to insert an interactive simulation for the student to explore.

Strong typing. Every variable and value may be declared and type-checked at compile time. Strong typing not only allows many programming errors to be caught at compile time but gives the Curl compiler the extra information it needs to produce efficient native code implementations.

Ambiguous types. In the absence of declarations, variables default to type `any`. A datum of type `any` is represented as a pair of words containing the run-time type of the datum and its value. The semantics of a Curl program are unchanged if all type declarations are eliminated, although its performance will suffer. This is the flip side of strong typing: providing types for each argument or variable can be a time consuming process which may not be worthwhile while prototyping new ideas or when efficiency is not of concern to the author.

Lexically scoped environment. Curl provides a structured name space whose bindings include variables, constants, types, and compilation hooks for arbitrary syntactic forms. The name space is instantiated as an *environment* which spans compile and run times, and which completely dictates the semantics that will be associated with source code during compilation. By carefully selecting which functions are made available in environments used to execute imported code, a considerable degree of safety can be achieved.

Procedural semantics. Procedures are first-class data, implemented using closures as necessary. Positional and keyword arguments are supported.

Objects. Types include classes, with multiple inheritance, arrays, and primitive types such as integers.

First-class types [11]. Types, including classes, are first-class objects: they are bound in the environment and obey consistent scoping rules; they may be arguments to or results from arbitrary computation both at compile and run-times. Having type information available at run time allows programs to examine and manipulate arbitrary objects without any pre-knowledge of their structure. This is particularly useful for applications like object inspectors or for building general purpose mechanism for

transporting objects from machine to machine.

Portability, Performance, and Safety

Efficient native representations. Representation conventions for values with declared types are tag-less, consistent with C and C++, yielding similar code efficiency.

Incremental compilation. Local compilation of source code to native code is the basis of Curl portability, safety, program encapsulation, and performance. Code that has been compiled may be cached locally in memory or on disk.

Other Features

Threads. Curl supports a simple platform-independent thread model. Threads are particularly useful in the "real-time" environment of the Web where the use of blocking input/output can lead to extremely frustrating delays when accessing foreign data.

Garbage Collection. By default, Curl objects are garbage collected when they are no longer used.

Types

Efficient implementation on contemporary processors, as well as easy interfacing to foreign software environments, dictates the use of conventional C/C++ representations of data. To provide typeless simplicity for simple scripting-level programming, Curl provides the ambiguous type `any` which may require the run-time representation of type information. Thus,

```
{let a:any=5
  {define {f x:int}:int {return {+x 3}}}
    {f a}}
```

will be compiled so that the ambiguously declared variable `a` carries a run-time type, while the formal parameter `x` to the integer procedure `f` does not. Passing `a` to `f` requires a cast, which involves a run-time type check after which the value portion of `a` is passed as an integer. To perform such casts efficiently, type information (when present) is separated from the representation of values. Because the storage for a value of type `any` is allocated by the compiler, casting a value to the ambiguous type never causes dynamic storage allocation.

Note that the omission of a type declaration for a variable means something different than in, for example, ML [12]. In ML the variable has a static type that the compiler must determine. In Curl, an omitted type is an under-specification and the actual type may change during program execution.

Since run-time representations are required for our extensible system of types, it is convenient to represent them as Curl objects and to expose them in the programming model. Thus, Curl types are simply values: they are named, passed, stored, and accessed identically to other Curl data. This treatment of types provides a natural and consistent semantics for type extension, for type portability (which reduces to the problem of object portability), and for extended type semantics. For example, in the following:

```
{let t1:type={array int}
     t2:type={list-of t1}
     x:t2
     ...}
```

`array` and `list-of` are procedures that generate types. This emancipation of types, while semantically appealing, presents certain challenges. It requires that compilation order be somewhat lazy, to accommodate circularities. Moreover, the expressibility of mutable type variables admits constructs whose efficient translation, or even whose meaning, is questionable. We currently forbid such constructs, requiring that types used in declarations be evaluable to compile-time constants.

Ambiguous types, as well as the object class hierarchy, yield a type lattice in which a value may have arbitrarily many types, only some of which are compile-time constants. The run-time type of a value `v` may be explored using `{typeof v}`, which returns the most specific type object appropriate to `v`, using run-time tags if necessary. Similarly, `{isa t v}` tests the membership of `v` in type `t`, again using the fully-disambiguated run-time type of `v`.

Objects

Modern programming practice demands support for objects, and their advantages are particularly compelling in the context of an extensible Web programming environment. Objects provide a natural implementation for the hierarchy of displayed images on a Web page, and they support incremental extension of previously defined behavior. In its pure form, however, an object-oriented programming model fails to meet the "gentle-slope" goal: the naive user cannot simply add a variable to his otherwise textual Web page, for example. A consequent subgoal of Curl is the graceful integration of object semantics into a lexically-scoped procedural programming model.

Most Curl types are classes. In the abstract, a Curl class provides a namespace whose scope includes objects of that class. Within a class, names can be bound to values, both variable and constant; the latter includes methods specific to the class. Instances of a class have a run-time representation containing slots for the values of variable data, as well as dispatch tables for efficiently invoking methods of the class. Our current implementation uses representations similar to those of C++.

Graphic Objects

Curl's object-oriented graphical user interface combines both high-level interfaces with the usual repertoire of low-level primitives. A graphic image is represented as a hierarchical tree of `Graphic` objects. Leaves of the tree are primitive `Graphic` objects that know how to draw themselves, usually after looking up the values of various properties (see below). Primitive `Graphic` objects include:

- *Lines and curves*. Properties control the color, width, dash style, and end treatment (e.g., none, arrowhead, ball, . . .).
- *Rectangles, polygons, ellipses*. Properties control the width and color of the border, and the stipple and color of the fill. Various 3D effects are also provided for the borders of rectangles.
- *Text*. Properties control the color, size, and font family as well as indicating whether the text should be bold or italic.
- *Multiline text*. Similar to text, but can display itself across multiple lines.
- *Color bitmaps*. Curl supports the usual collection of image formats (GIF, JPEG, etc.).
- *Glue*. Glue has no visible representation but provides some parameterized dimensional flexibility that is useful in the formatting operations provided by Boxes (see below).

The basic graphic building block is the Box, which serves as the parent to a collection of primitive `Graphic` objects and instances of other Boxes. The displayable hierarchy consists of a tree (or graph) of such objects.

In addition to providing the abstraction mechanism for building complex images, various types of Boxes also control the positioning of their children using dimension information the children supply. Current formatting templates include:

- *Hboxes and vboxes*. These are one-dimensional formatters that create simple horizontal or vertical arrangements of their children, lining up their baselines or margins. As in TeX, the relative allocation of whitespace is controlled by the elasticity of any glue objects that have been added as children.
- *Tables*. A two-dimensional formatter that aligns the margins and baselines of its children on a Cartesian grid. The row and column of each child is specified as it is added to the table template; a child can span multiple rows and/or columns.
- *Paragraphs*. Children are positioned left-to-right, top-to-bottom. Children on the same line have their baselines aligned and the interline spacing is chosen to accommodate the tallest child on a line.
- *Canvases*. No explicit formatting is provided; each child is given a position as it's added to the canvas template.

This set of formatting primitives is sufficient to duplicate the effects of most text formatters and browsers with little effort on the part of the Curl user.

Much of the flexibility of Boxes comes from the use of properties to control the rendering of primitive objects. A property is a (name,value) binding and each `Graphic` object has an associated list of properties. When the value of a property is required, an upward search of the template/instance tree is performed, starting with the current object. Thus, properties can be viewed as a dynamically bound environment implemented using a deep binding mechanism. So, as in HTML, many of the rendering choices--color, font, etc.--are not made by individual templates but are instead inherited from their parents through their property bindings.

Notification of a change in a property's value is propagated through the tree which may, in turn, cause various reformatting operations to take place. If some `Graphic` object determines that its image has changed, it can request that it be re-rendered. Since there is sharing of objects within the tree, a re-render request may in fact cause more than one location on the output device to be redrawn. Curl uses the invalidate/repaint protocol used by many GUI frameworks, but users need not concern themselves with this machinery unless they want to.

As in Java, the dispatching of events (mouse motion, keystrokes, and so on) is controlled by the tree where the event handler for the visually topmost `Graphic` object that encloses the event location is invoked to deal with the event. Handlers can pass the event up the tree if they don't wish to handle the event themselves.

Naming and Environments

The resolution of names outside of classes, and its graceful integration with names within the class hierarchy, again involves the exposure of compilation machinery to potentially introspective Curl programs. Our goal was to provide a simple, structured, lexically-scoped environment for binding variables, types, constants, and language extensions.

Classes are bound within the environment, neatly embedding the hierarchy of class names as subtrees of the namespace. Name resolution is kept simple and unambiguous by requiring the syntax `obj.x` to access `x` from an object's class; the unqualified name `x` always references the innermost lexical binding. Thus, as shown in Example 2, the reference `x` within the method `m` gets `3`, while `self.x` gets `4`.

Note that types may be generated anonymously via `class`. Names of types derive solely from bindings in the environment, where `c` is the name of a type in the example. The `define-class` form builds a class and enters it into the current (lexically enclosing) environment.

Safety, Storage, and Encapsulation

The ability to import code from untrusted sources demands mechanisms for circumscribing the behavior of imported programs at various levels, providing "safe" execution relative to some criterion. The appropriate choice of safety criteria may vary depending on its purpose or source, or on the priorities of the user. A performance-critical JPEG decoder from a cryptographically-secured source, for example, might be executed unencumbered by run-time overhead; an applet from a strange Web page, however, might warrant carefully encapsulated execution. This variation engenders important subproblems:

- Policy issues which dictate the safety criterion
- Enforcement mechanisms sufficiently flexible to realize various performance/safety tradeoffs

We view the first as a subject for continuing research, supported in Curl by its attention to the second.

Curl, as a language, has a type system sufficiently high-level that Curl semantic guarantees can be maintained despite buggy or malignant code via a combination of compile-time and run-time type checks. This is exactly how Java provides this guarantee. These guarantees are the default for normal execution of untrusted code, but entail performance costs and functional restrictions that may be undesirable in high-performance code. For example, stack allocation or allocation in areas that are then reused may provide significant performance advantages. These unsafe mechanisms are also part of the Curl language but are not available in the environment used by untrusted code.

Since the compilation of Curl forms involves invoking compilation machinery attached to those forms in the compilation environment, the semantics of a Curl program are entirely derived from the environment in which it is compiled. A program compiled in an environment with no bindings for `set` or `open-file`, for example, has no access to these primitives. As all imported code is locally compiled, the tailoring of compilation environments provides an airtight mechanism for enforcement of an arbitrary variety of safety policies.

An untrusted applet from an interesting Web page might, for example, offer the user a desktop playpen in which a local simulation environment can be configured. In order to provide for the storage of the applet's state between visits to that page, local file system access is necessary; however, unconstrained access to the local file system presents a security flaw. Such needs can be safely accommodated, however, by providing a limited-access primitive which allows an applet to save

```
{let x:int=3                    | Initialize local variable
    c:type={class {}            | Define local class c
          x:int
          {define {m}:int {return {+ x self.x}}}}
    y:c={new c}                 | Allocate an instance of c
    {set y.x 4}                 | Assign to instance variable
    {y.m}}                      | invoke method & return value
```

its state in a particular file from a user-mediated path, and to subsequently restore it transparently. An untrusted program might request a `savelet` object which supports such limited access to a single applet-specific file, and be prevented (by the restricted compilation environment) from arbitrary file accesses.

The low-level requirement for any security policy is freedom from stray memory accesses (minimally, from stray writes) and stray system calls. For a safe subset of Curl, one that does not allow stack allocation and other unsafe operations, this guarantee is provided by the above mentioned compilation machinery. For arbitrary Curl programs, an alternative to simply trusting that some code is correct is to use "sandboxing" [17]. In sandboxing, the compiler inserts a few extra instructions around writes and procedure calls to make sure that writes only occur to areas in memory that the compiler knows are okay, and that calls are only made to procedures under the compiler's control. A system environment that provided sandboxing would be an attractive target for Curl.

Comparison With Other Languages

Along with formatting and markup languages (like TeX and HTML), Curl shares the property that its input defaults to a textual document, and that its explicit output is itself a viewable image of that document. Markup languages like HTML don't address general purpose programming; to the extent that TeX does, it addresses it poorly.

Through the `object` tag, HTML allows an arbitrary external computation to be invoked from an HTML document and its result to be embedded in the document. Of course, the rest of the document cannot interact with the computation. In Curl, by contrast, the markups are expressions in the underlying language and can have arbitrary interactions with the document.

Tcl/Tk provides a scripting language and widget set that allows user interfaces to be constructed easily and integrated with a program. Curl procedures with types defaulting to `any` also provide this functionality but offer several advantages. First, Tcl is very inefficient because it represents everything as a string at the semantic level, while Curl's representations are based on objects and can be compiled to efficient code. Second, in order to provide new widgets in Tcl/Tk, one has to write code in C. Curl provides a uniform interface between code at all levels.

Syntactically, Curl resembles LISP and its dialects. We rely on the syntactic extensibility of prefix notation provided by LISP and extend it to include extensible text formatting as part of the language. Curl also differs from LISP in that it is, at its foundation, a statically-typed object-oriented language with non-tagged basic data representations. The part of Curl that is used like a conventional programming language most resembles Dylan [10], before it was changed to use C-like syntax.

The object semantics of Curl are similar to those of Java and C++. Unlike Java, Curl embeds objects within a lexically-scoped procedural programming model, it supports multiple inheritance, and it allows trusted code to exploit the semantic and efficiency advantages of unsafe operations. Unlike C++, Curl supports type safety.

Curl Implementations

Our Curl implementation contains the following components, all written in Curl:

- Platform-independent GUI.
- An incremental compiler that produces native code for x86 and Sparc. Work is underway on producing code for the Java Virtual Machine.
- An object inspector and symbolic debugger for code.
- A browser for viewing Curl programs.

The basic infrastructure code, such as the incremental compiler, are built into a static program image using a bootstrap compiler that compiles the non-formatting portion of Curl into C. This generated C code is linked with some hand-written C code that supports bootstrapping the Curl type system and kernel. Currently supported platforms are:

- PC running Windows NT, Windows 95, or Linux
- Sparc-based machines running SunOS or Solaris

Curl is available as a stand-alone executable for these platforms with plans for a compatible subset of Curl on the Java Virtual Machine in the near future. Interested readers are referred to the Curl Web site (*http://cag-www.lcs.mit.edu/curl*) for up-to-date information status and availability of the various Curl implementations.

Client versus Server-Side Processing

Because Curl is a general purpose, efficient programming language, it can be used to perform both client-side and server-side processing. In fact, when both the client and server are Curl programs, functionality can be adjusted dynamically between the client and server. We currently use a Curl program on our server to distribute Curl documents by translating them, on the fly, to HTML. This mechanism allows the browsing of Curl documentation without downloading the Curl display engine and compiler. Of course, many things that can be expressed in Curl cannot be translated into HTML, but could be translated into code for the Java virtual machine. Such code would run much less efficiently than our native Curl implementation but would allow a Curl author/programmer to rely on the fact that even the most naive user would be able to look at at her document.

Summary

One of the weak points of the Web today is that there is a dichotomy between programs and documents, and also between programmers and authors. This dichotomy is a result of the lack of a linguistic tool with sufficient generality. Increasing the active content of a document should be as simple as making additional regions in the document show the result of a computation, and supplying code to compute what should be displayed in those regions. It should also not be necessary to rewrite parts of a document in a different language because it turned out that some part of the code ran too slowly. The Curl language and implementation show that it is possible to meet these goals.

The goal of integrating an authoring/programming language and environment with a gentle slope is a challenging one. There are many places where the most natural or useful syntax and semantics are in conflict. We have tried to resolve these conflicts in a tasteful way. We also had to choose a set of object semantics, a subject about which there is great controversy. The object semantics we have implemented have little impact on the overall system except that we insisted on efficient representations and low dispatch overhead.

Producing an interactive program/document is much easier when the appropriate level of abstraction can be used for all components without having to cross language and representation barriers. It is a challenge to provide high-performance and flexibility but our initial results are encouraging.

See *http://home.netscape.com/comprod/products/navigator/version_3.0/building_blocks/jscript/index.html*.

Gosling, Joy, Steele. *The Java Language Specification*, Addison-Wesley. August 1996.

Java white papers at *http://java.sun.com/nav/read/whitepapers.html*.

Clinger, William, and Jonathan Rees (editors). *The Revised Report on the Algorithmic Language Scheme*, Lisp Pointer IV(3): 1-55, July-September 1991.

Ousterhout, John K. *Tcl and the Tk Toolkit*, Addison-Wesley, 1993.

World Wide Web Consortium. *http://w3.org/*.

Berners-Lee, T., and D. Connolly. *Hypertext Markup Language: A Representation of Textual Information and Metainformation for Retrieval and Interchange*, CERN and Atrium Technology Inc. July 1993.

Berners-Lee, T., and D. Connolly. RFC 1866: Hypertext Markup Language--2.0, Nov 3, 1995.

Knuth, Donald E. *The Texbook: A Complete Guide to Computer Typesetting With TeX*, Addison-Wesley. April 1, 1988.

Apple Computer, *Dylan Reference Manual*, Apple Computer Inc., Cupertino, California, September 29, 1995. See also *http://www.cambridge.apple.com*.

Cardelli, L. *A Polymorphic Lambda Calculus with Type:Type*, Digital Equipment Corp., Systems Research Center, 1986.

Milner, Robin, Mads Tofte, and Robert Harper. *The Definition of Standard ML*, MIT Press, Cambridge, 1990.

Guy Steele, Jr. *Common Lisp,* Second Edition, Digital Press 1990.

Moon, David, et al, *LISP Machine Manual,* Fifth Edition, MIT Artificial Intelligense Laboratory, January 1983.

Bobrow, Daniel, et al. *A Common Lisp Object System Specification, Lisp and Symbolic Computation I*, January 1989.

Stroustrup, Bjarne. *The C++ Programming Language*, Addison Wesley, 1991.

Wahbe, Robert, Steven Lucco, Thomas E. Anderson, Susan L. Graham. "Efficient Software-Based Fault Isolation," *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*. December, 1993.

Acknowledgments

The Curl project is supported by the Information Technology Office of the Defense Advanced Research Projects Agency as part of its Intelligent Collaboration and Visualization program.

About the Authors

All the authors can be contacted at:
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA, 02139

General questions and comments can be sent to *curl@lcs.mit.edu*; email addresses for each author are given below.

Mat Hostetter (*mat@lcs.mit.edu*) is an M.Eng candidate in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. His current research interests include run-time code generation, CPU emulation, and high-performance compilers.

David Kranz (*kranz@lcs.mit.edu*) is a Principal Research Scientist at the MIT Laboratory for Computer Science. His research interests are in programming language design and implementation for parallel computing. Kranz received a B.A. from Swarthmore. While earning a Ph.D. at Yale, he worked on high-performance compilers for Scheme and applicative languages.

Cotton Seed (*cottons@lcs.mit.edu*) is a member of the research staff at the MIT Laboratory for Computer Science.

Chris Terman (*cjt@mit.edu*) is a Research Scientist at the MIT Laboratory for Computer Science. Chris has worked in both academic and industrial settings on a variety of projects including VLSI architectures and design methodologies, computer-aided design tools, language and compiler technology, and Web-based educational tools. He received a B.A. in Physics from Wesleyan University and S.M., E.E., and Ph.D. degrees in Computer Science from MIT.

Stephen Ward (*ward@mit.edu*) is a Professor in the Electrical Engineering and Computer Science Department at the Massachusetts Institute of Technology.

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☑ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.